CDR/FCDR Writer/Reader Usage

Tom Block, Sabine Embacher

Brockmann Consult GmbH

8/27/2019

FIDUCEO has received funding from the European Union's Horizon 2020 Programme for Research and Innovation, under Grant Agreement no. 638822

1 Introduction

This document describes the usage of the FIDUCEO tools for reading and writing CDR and FCDR data files.

This set of tools is programmed in Python and has been tested using Python versions 3.5 and 3.6 on Linux and Windows operating systems.

The software is available at Github (https://github.com/FIDUCEO/FCDRTools/releases/tag/2_0_1_RELEASE).

1.1 Version Control

Version	Reason	Reviewer	Date of Issue
1.0	Initial version		
1.1	Update to release 1.1.0	Sabine Embacher	08.09.2017
1.1	Update to release 1.1.3		21.02.2018
1.2	Update to release 1.1.4		27.04.2018
1.3	Update to release 1.1.5, added CDR specification		04.07.2018
2.0	Update to release 2.0.0		24.08.2018
2.0.1	Update to release 2.0.0		27.08.2019

1.2 Applicable and Reference Documents

The following documents are applicable (AD) or references (RD) in this manual:

RD-1 File format specification CDR_FCDR_File_Format_Specification_v2_0_1

1.3 Glossary

CDR	Climate Data Record
FCDR	Fundamental Climate Data Record

2 Common Elements

This section lists information that is common when writing CDR and FCDR files.

2.1 File format

CDR and FCDR writer store the data in NetCDF4 compressed format following the file format specification document [RD-1].

2.2 Global Metadata

A number of mandatory global product metadata fields has to be supplied – if omitted, the writing will fail.

```
dataset.attrs["institution"] = "Brockmann Consult GmbH"
dataset.attrs["title"] = "His Majesty!"
dataset.attrs["source"] = "fake data"
dataset.attrs["history"] = "none"
dataset.attrs["references"] = "CDF FCDR File Spec"
dataset.attrs["comment"] = "just to show ho things are intended to be used"
```

2.3 Scaled Integer Data

When writing variables to NetCDF it is often beneficial to store the data in an integer data format using a scaling factor and an offset as attributes to allow regenerating the original floating point value.

When using inappropriate scaling factor and offset values, this approach can easily lead to numerical over- or underflows which are not detected or handled by the underlying xArray implementation of the CDR/FCDR writer.

To check for possible data errors due to numerical range problems, the FCDR Tools offer a utility class that verifies correct scaling results.

```
DataUtility.check_scaling_ranges(variable)
```

This method takes as only argument a Variable. The variable has to be equipped with encoding information containing target data type, fill value, scaling factor and offset. Example:

```
variable.encoding = dict([('dtype', np.int8), ('_FillValue', -127), ('scale_factor', 0.001),
('add_offset', 12)])
```

In addition, the checking utility takes into account possible value range restrictions introduced by the CF-conforming attributes “valid_min” and “valid_max”, if present.

When issues during scaling are detected, the method raises a ValueError. When everything seems to be working correctly, the method just returns.

2.4 Writing options

The write method of the CDR and FCDR writer takes two mandatory arguments, these are the filled dataset and the target path name.

```
writer.write(dataset, filepath)
```

In addition, the writer allows to fine tune the data compression level using a third parameter. Compression levels range from 0 to 9, the default value (when omitting the parameter) is 5.

```
writer.write(dataset, filepath, compression_level=7)
```

The writer allows to select the behaviour when trying to write to a location where already a file exists. This can be set by using the overwrite parameter. When set to true, the writer overwrites the already existing file, when set to false, the writer raises an exception. The default value is false.

```
writer.write(dataset, filepath, overwrite=true)
```

3 FCDRWriter

The FCDRWriter tool serves as a means to ensure that the data written to FCDR files follows the file specification. It achieves this by offering pre-defined data product templates for each input sensor/data type which follow the specification. The users of the writer merely have to fill in the data for the variables, the formatting is applied automatically.

3.1 Usage

The general usage pattern for the writer is depicted below:

Request a template dataset from the writer. Parameters are the sensor name and the number of scanlines in the data product. The width of the scans is determined by the sensor type automatically.

```
dataset = FCDRWriter.createTemplateFull("AVHRR", 12835)
```

All variables of the template are initialized with the per-variable fill value. The user has to supply real data only on locations where data is available.

```
dataset.variables["Ch1_Bt"].data[23, 44] = 0.456
```

Fill in the common global metadata fields as listed in chapter 2.1.

When all data has been written to the template it can be stored to the HD

```
writer.write(dataset, "D:\\Satellite\\TEMP\\AVHRR_fcdr_full.nc")
```

During the writing process, the FCDRWriter performs a number of operations in the background, predominantly:

- Compressing floating point data variables to scaled integer NetCDF variables where configured (see also 2.3)
- Process flag-mapping from sensor specific to global flag variables

3.2 Templates

A template or template dataset is a data structure that mirror the target file content as structured tree of objects in the RAM. It consist of all variables, data arrays, variable attributes and global attributes that comprise the target file, most of the attributes are set to their final values, all data arrays initialized to the appropriate fill value.

The FCDRWriter offers two method to request data templates, one for the EASY and one for the FULL FCDR data format; both methods accept the same parameters:

```
createTemplateEasy(sensorType, height, srf_size=None, corr_dx=None, corr_dy=None, lut_size=None)
createTemplateFull(sensorType, height)
```

The ***sensorType*** parameter defines the sensor, valid inputs are:

- AMSUB
- AVHRR
- HIRS2
- HIRS3
- HIRS4
- MHSS
- MVIRI

The ***height*** parameter defines the height of the data-arrays, i.e. the number of scan lines. The data-array width is already defined by the sensor parameter.

The optional parameters for the EASY FCDR template allow to adapt the format:

srf_size: size of the spectral response function related variables. If set, this parameter overwrites the default values to fine-tune the dataset content. The default values are sensor dependent:

Sensor	Default SRF length
AVHRR	5902
HIRS2	102
HIRS3	191
HIRS4	2751
MVIRI	1011

corr_dx, corr_dy: define the size of across and along track correlation coefficients vectors. If one of these is not set, the corresponding variables are not present in the template.

lut_size: defines the dimension of the per-channel lookup tables to perform BT/radiance conversion. If not set, the lookup tables will not be present in the template

3.3 File Name Generation

To ensure a uniform file naming convention, the FCDRWriter offers helper methods to create file names following the naming patterns defined for the FIDUCEO project.

```
create file name FCDR easy(sensor, platform, start, end, version)
create file name FCDR full(sensor, platform, start, end, version)
```

Both methods accept the same set of parameters:

- sensor: the satellite sensor name
- platform: the satellite platform name
- start: sensing start date and time, type *datetime*. The code assumes the datetime object to use UTC time, no time-zone adjustments are performed.
- end: sensing end date and time, type *datetime*. The code assumes the datetime object to use UTC time, no time-zone adjustments are performed.

- version: processor version of type string, format “xx.x”

The methods return a file name following the FIDUCEO file naming conventions.

4 FCDR Reader

The FCDR Reader tool supports third party code to handle FCDR datasets. The FCDR files are stored in standard NetCDF file format, so a large number of software components are capable of reading the plain data files. The specific FIDUCEO FCDR reader module adds a convenient functionality on top of that. The FULL FCDR format contains so called “virtual variables”, which are variables that can be calculated from the data contained in the file. This affects mostly the sensitivity coefficients. To save storage space, the virtual variables in FULL FCDR files contain only the mathematical expression to calculate the per-pixel values.

The FCDR reader can apply the mathematical formulae and automatically expand the virtual variables to full raster variables. It does this “on-demand”, so only the data requested by the user is calculated, to save space and processing resources.

The FCDR Reader exposes one method that reads a FCDR file into a XArray dataset object:

```
def read(file_str, drop_variables_str=None, decode_cf=True, decode_times=True, engine_str=None)
```

- file_str: The FCDR netCDF file path
- drop_variables: (optional) list of variable names to drop (i.e. not to load)
- decode_cf: Whether to decode CF attributes and coordinate variables
- decode_times: Whether to decode time information (convert time coordinates to ``datetime`` objects)
- engine_str: Optional netCDF engine name

These parameters are identical to the ones used by the XArray open_dataset method.

5 CDRWriter

The CDRWriter tool serves as a means to ensure that the data written to CDR files follows the file specification. It achieves this by offering pre-defined data product templates for each CDR data type which follow the specification. The users of the writer merely have to fill in the data for the variables, the formatting is applied automatically.

5.1 Usage

The general usage pattern for the writer is depicted below:

Request a template dataset from the writer. Parameters are the sensor name and the width and height of the data product.

```
dataset = CDRWriter.createTemplate("AOT", 409, 12835)
```

When requesting a template for an ensemble-based CDR an additional parameter “num_samples” has to be supplied which defines the number of statistical samples that constitute the ensemble dataset.

```
dataset = CDRWriter.createTemplate("SST_ENSEMBLE", 409, 12613, 14)
```

All variables of the template are initialized with the per-variable fill value. The user has to supply real data only on locations where data is available.

```
dataset.variables["u_independent_aot"].data[23, 44] = 0.026
```

Fill in the common global metadata fields as listed in chapter 2.1.

In addition, CDR specific metadata has to be supplied:

```
dataset.attrs["source"] = "The things used as input"
dataset.attrs["auxiliary_data"] = "additional data used. Set to None if not"
dataset.attrs["configuration"] = "whatever was used to configure the processing"
dataset.attrs["time coverage start"] = "20180205T235959Z"
dataset.attrs["time coverage end"] = "20180206T011421Z"
dataset.attrs["time coverage duration"] = "P1H15M26S"
dataset.attrs["time_coverage_resolution"] = "P1S"
```

Please note that the fields “time_coverage_duration” and “time_coverage_resolution” are supposed to be in ISO8601 format to follow the CCI format specifications (https://en.wikipedia.org/wiki/ISO_8601).

When writing a gridded data product the following L3 specific metadata fields have to be supplied:

```
dataset.attrs["geospatial_lat_units"] = "degrees_north"
dataset.attrs["geospatial_lon_units"] = "degrees_east"
dataset.attrs["geospatial lat resolution"] = "0.1"
dataset.attrs["geospatial_lon_resolution"] = "0.1"
```

When all data has been written to the template it can be stored to the HD

```
writer.write(dataset, "D:\\Satellite\\TEMP\\AOT_cdr.nc")
```

During the writing process, the CDRWriter performs a number of operations in the background, predominantly:

- Compressing floating point data variables to scaled integer NetCDF variables where configured (see also 2.3)

5.2 Templates

A template or template dataset is a data structure that mirror the target file content as structured tree of objects in the RAM. It consist of all variables, data arrays, variable attributes and global attributes that comprise the target file, most of the attributes are set to their final values, all data arrays initialized to the appropriate fill value.

The CDRWriter offers one method to request data templates for the CDR data format:

```
createTemplate (data_type, width, height)
```

The ***data_type*** parameter defines the target data type structure, valid inputs are:

- ALBEDO
- AOT
- SST
- SST_ENSEMBLE
- UTH

The ***width*** and ***height*** parameters defines the dimension of the data-arrays, i.e. the number of scan lines and the length of a scanline (or geographical axes for gridded data products).

5.3 File Name Generation

To ensure a uniform file naming convention, the CDRWriter offers a helper methods to create file names following the naming pattern defined for the FIDUCEO project.

```
create file name CDR(data type, sensor, platform, start, end, type, version)
```

Both methods accept the same set of parameters:

- data_type: the geophysical data type (SST, UTH, etc.)
- sensor: the satellite sensor name
- platform: the satellite platform name
- start: sensing start date and time, type *datetime*. The code assumes the datetime object to use UTC time, no time-zone adjustments are performed.
- end: sensing end date and time, type *datetime*. The code assumes the datetime object to use UTC time, no time-zone adjustments are performed.
- type: the data storage type (L2, L3, ENSEMBLE)

- version: processor version of type string, format “xx.x”

The methods return a file name following the FIDUCEO file naming conventions.

6 CDR Reader

The CDR data is stored in plain NetCDF format, there is no specific reader software necessary to handle these data.